

# A Hands-On Java PathFinder Tutorial

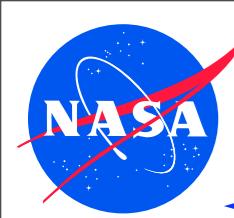
## ICSE 2013

## Part 1: Introduction

Neha Rungta  
SGT / NASA Ames Research Center  
[<neha.s.rungta@nasa.gov>](mailto:<neha.s.rungta@nasa.gov>)

Peter C. Mehlitz  
SGT / NASA Ames Research Center  
[<peter.c.mehlitz@nasa.gov>](mailto:<peter.c.mehlitz@nasa.gov>)

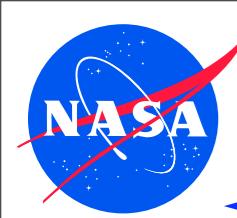
Willem Visser  
University of Stellenbosch  
[<wvisser@cs.sun.ac.za>](mailto:<wvisser@cs.sun.ac.za>)



# JPF-Tutorial: Where Are the Slides?



- ◆ <http://babelfish.arc.nasa.gov/trac/jpf/raw-attachment/wiki/presentations/start/ICSE-2013-Tutorial.pdf>
- ◆ or go to <http://babelfish.arc.nasa.gov/trac/jpf/wiki/presentations/start>, scroll down to the “Attachments” section, expand and *click on the download icon trailing the filename*

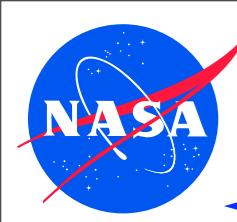


# JPF Tutorial: Roadmap

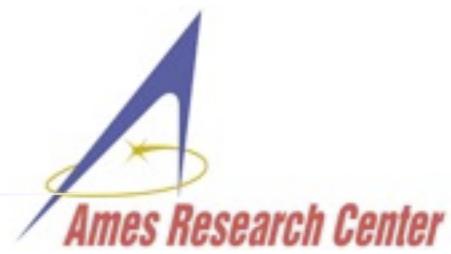


- ◆ 1. Basics
  - what is JPF
  - core concepts and extension mechanisms45min
  
- ◆ 2. Install and Run JPF
  - build & test
  - run basic examples (Rand, Racer)1h
  
- ~ Break ~30min
  
- ◆ 3. Using and Extending JPF
  - RangeChecker (safety property listener for field values)
  - StopWatch (control flow enumeration with attributes & listeners)2h

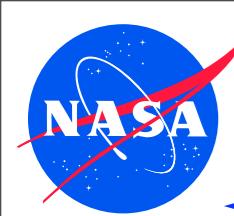




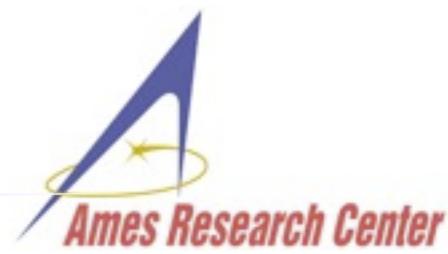
# 1. Basics



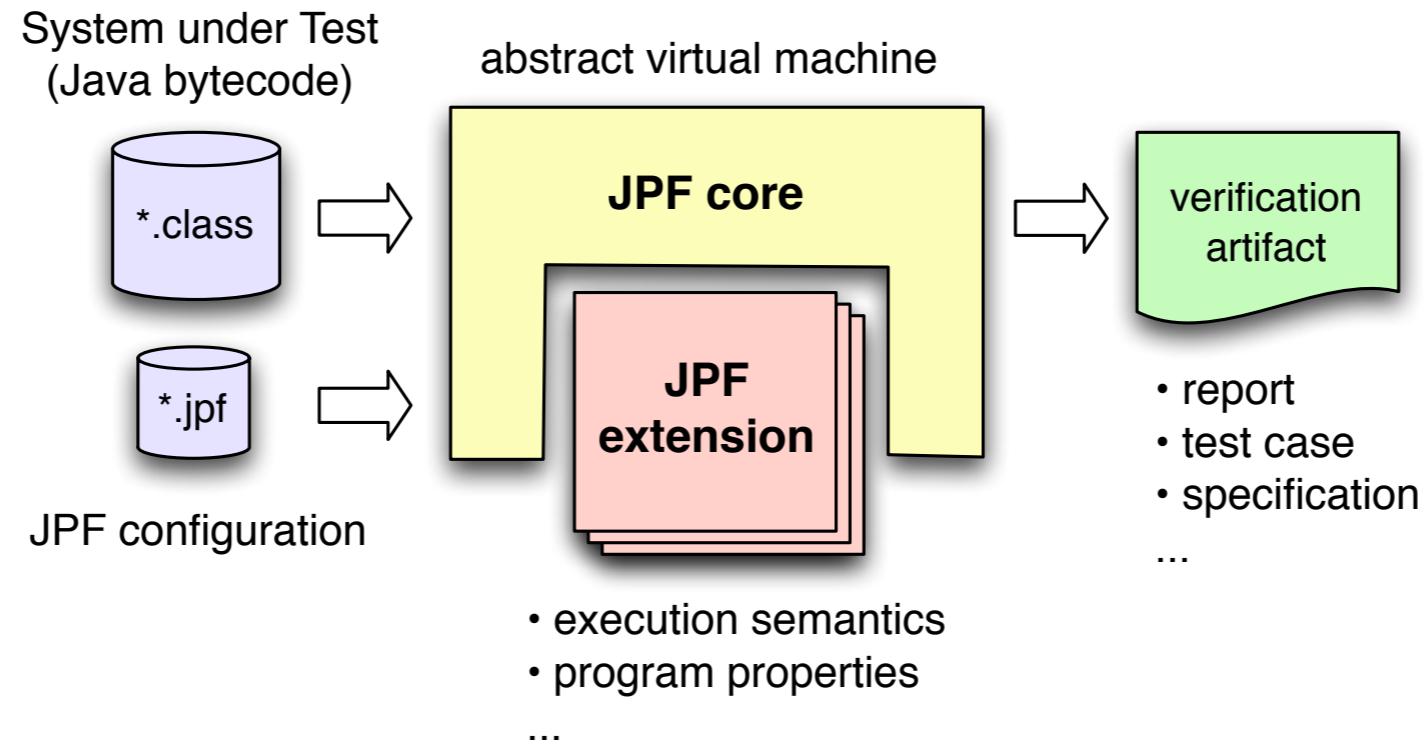
- ◆ the inevitable “What is JPF”
- ◆ concepts and mechanisms that are used in hands-on examples:
  - ChoiceGenerators
  - Listeners
  - Attributes
  - .. NativePeers, InstructionFactories
- ◆ JPF project layout
- ◆ JPF configuration



# Basics: JPF - What is it?



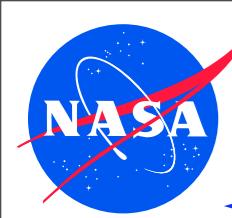
- ◆ a model checker? a virtual machine? ..



- ◆ ..and the answer is: both, and more - it depends on you
- ◆ *not* a monolithic, black box tool

No “one size fits all” - Extensibility is Paramount

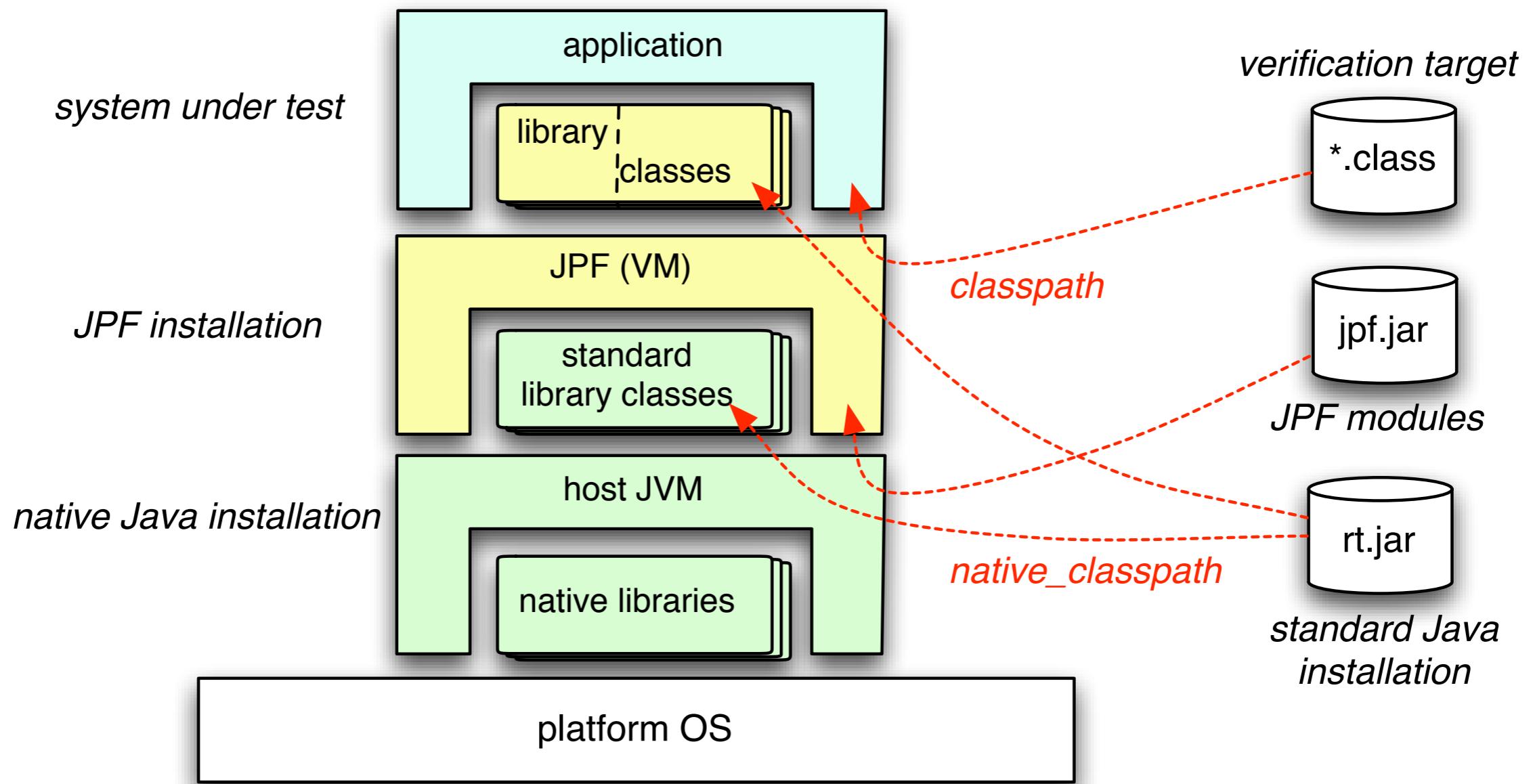
- ◆ the quest of today: learn what is in the toolbox to find out how you can adapt JPF to *your* needs

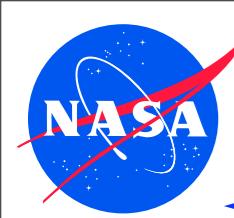


# Basics: a VM running inside a VM

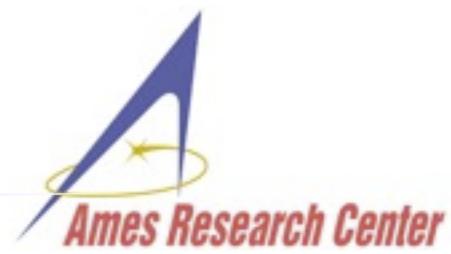


- ◆ main stumbling block is recursive nature of JPF
- ◆ verified Java program is executed by JPF, which is a virtual machine implemented in Java, i.e. runs on top of a host JVM
- ◆ need for two separate classpaths (host VM and JPF)

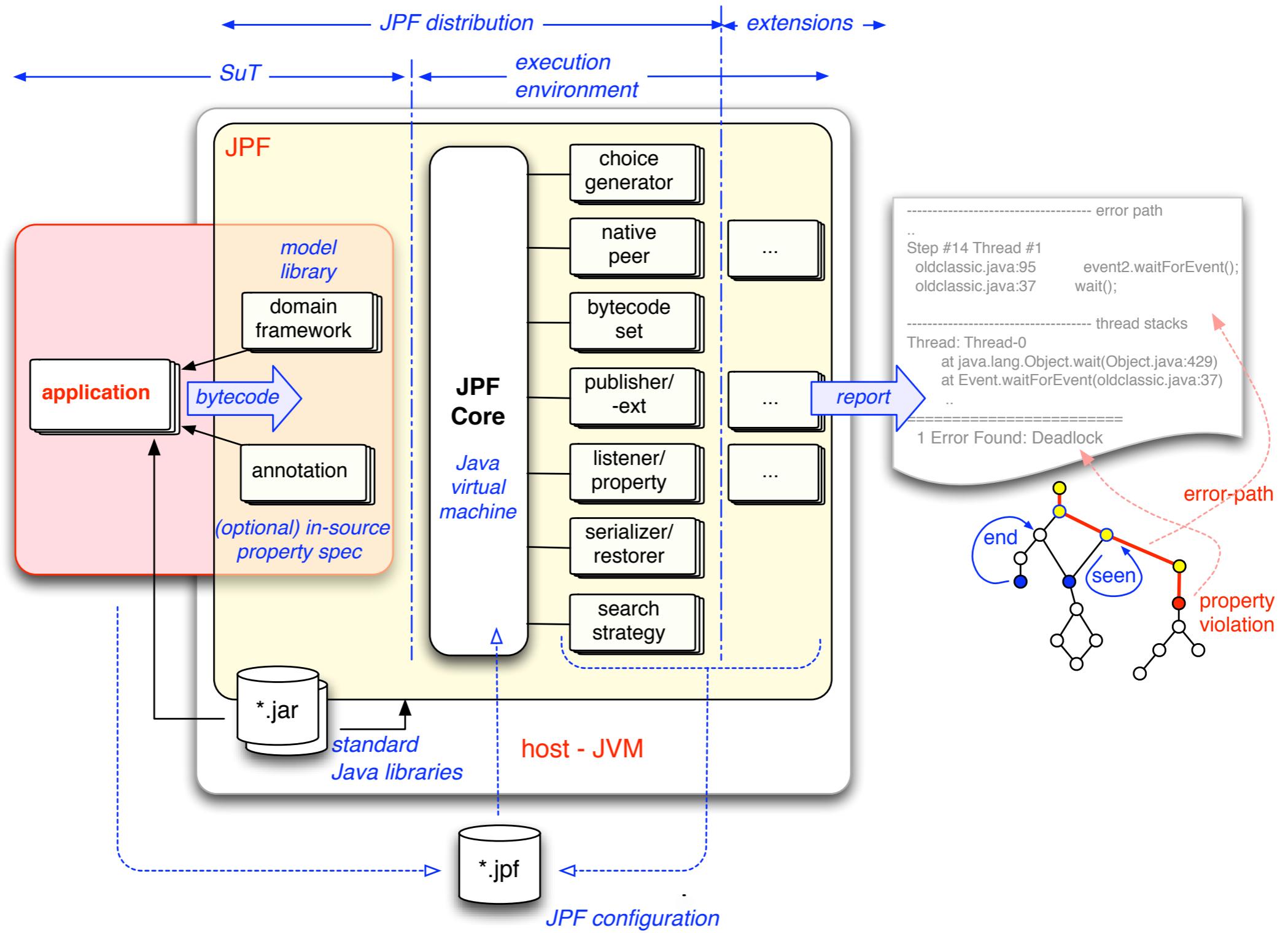


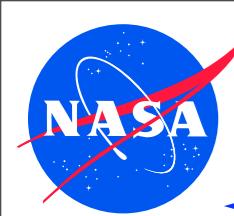


# Basics: JPF Extensibility



- ◆ JPF has many extension points, jpf-core is just the glue

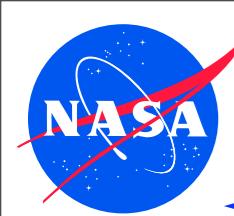




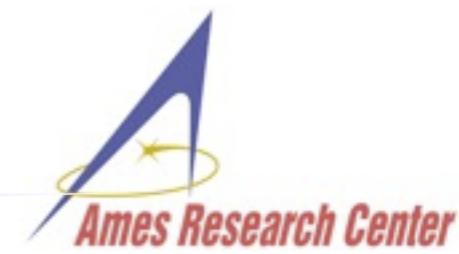
# Basics: Concepts and Extension Mechanisms



- ◆ top level design
  - ◆ ChoiceGenerators - Transitions and States
  - ◆ VM and Search Listeners - the JPF “plugins” *used in tutorial*
  - ◆ Object Attributes - data flow
- 
- ◆ Native Peers - Model Java Interface (MJI)
  - ◆ Instruction Factories - execution semantics
  - ◆ ... and many more (Serializers, Publishers, PublisherExtensions, ...)



# Basics: JPF Toplevel Design



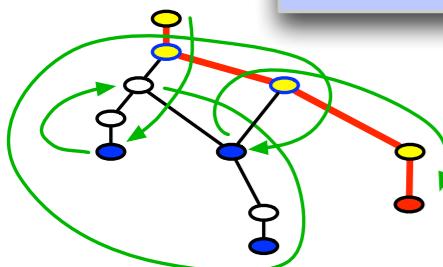
VM driver Search

search.class=...

gov.nasa.jpf.search

Search  
JVM vm  
search ()

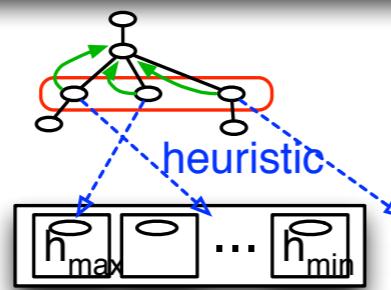
DFSearch  
search ()



gov.nasa.jpf.search.heuristic

HeuristicSearch  
search ()

BFSHeuristic  
search ()



JPF builder

gov.nasa.jpf

JPF  
search, vm, config  
run ()

Config

JVM state producer

vm.class=...

gov.nasa.jpf.jvm

JVM  
forward ()  
backtrack ()  
restoreState ()

Heap

StaticArea

ElementInfo

Fields

Monitor

ClassInfo

MethodInfo

FieldInfo

type + code  
mgnt

object  
model

SystemState

initializeNextTransition()  
executeNextTransition()

ThreadInfo

executeTransition ()  
executeInstruction ()

StackFrame

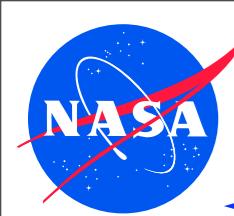
push (), pop() ...

bytecode execution

gov.nasa.jpf.jvm.bytecode

Instruction  
execute ()

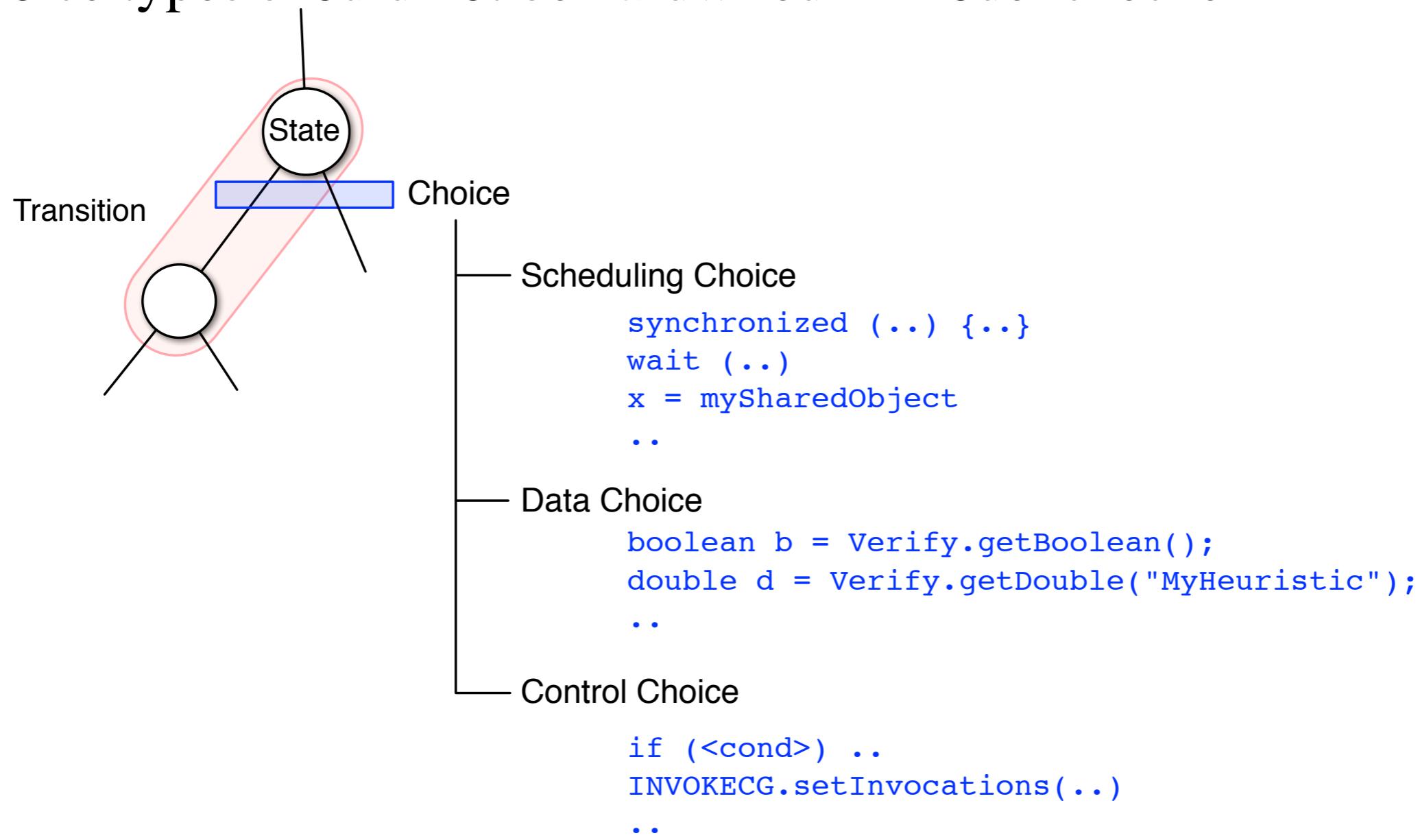
...

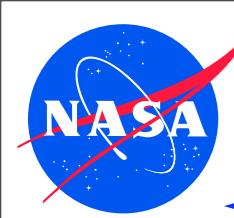


# Basics: Choice Generators



- ◆ model checker needs choices to explore state space
- ◆ there are many potential types of choices (scheduling, data, ..)
- ◆ some types can't be automatically enumerated (e.g. double)
- ◆ choice types should not be hardwired in model checker



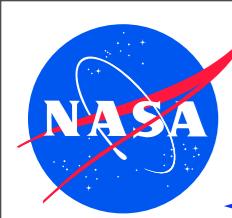


# Basics: ChoiceGenerator Code

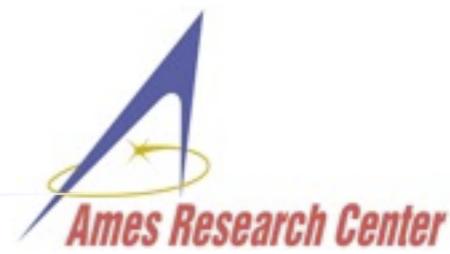


- ◆ enumerator object for type specific choice values

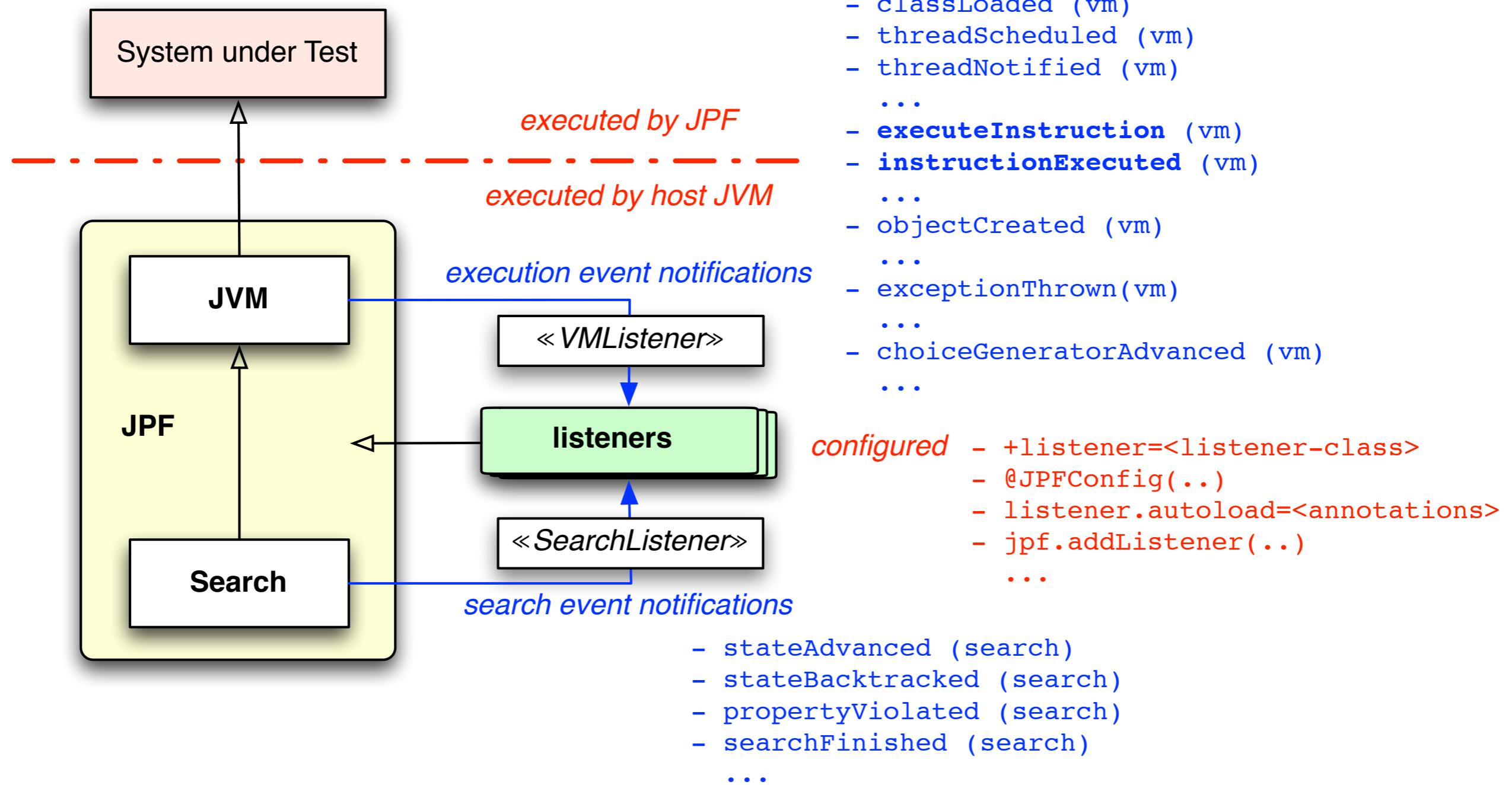
```
public class DoubleThresholdGenerator extends DoubleChoiceGenerator {  
    ...  
  
    public boolean hasMoreChoices () {  
        // are we done yet  
        ...  
    }  
  
    public void advance () {  
        // position on next choice  
        ...  
    }  
  
    public Double getNextChoice () {  
        // return choice value at current position  
    }  
  
    ...
```

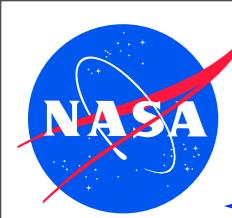


# Basics: Listeners - the JPF plugins



- ◆ executed at host VM level, can access all exported JPF features
- ◆ includes Search and VM events, both high- and low-level
- ◆ dynamically configured at runtime

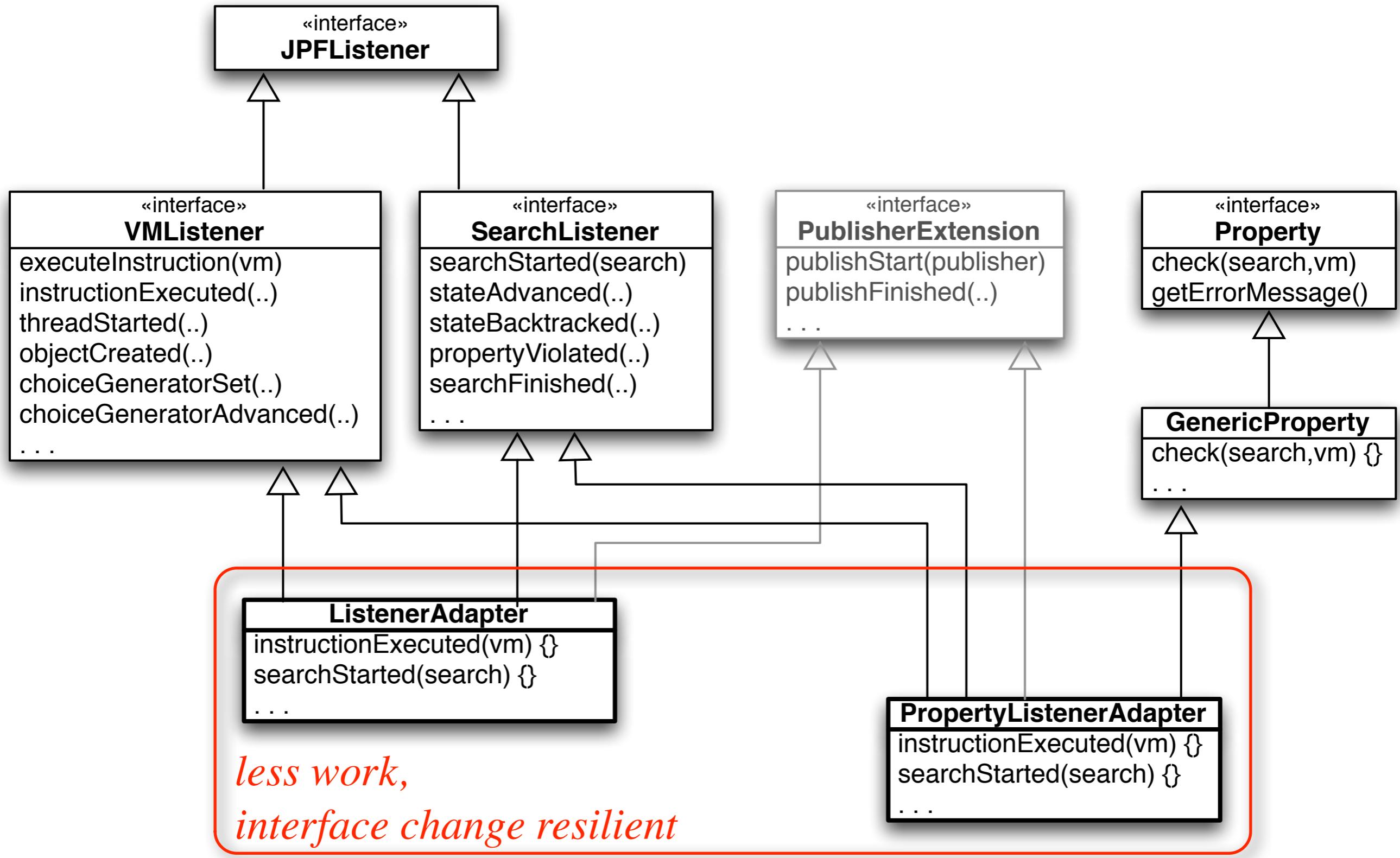


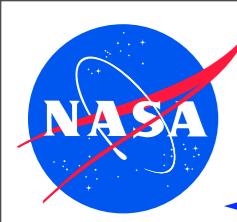


# Basics: Listener Implementation

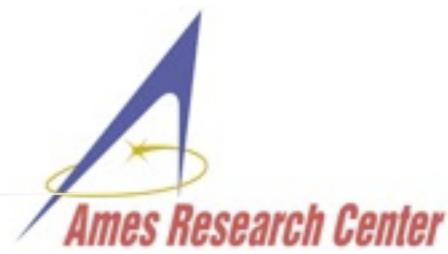


- ◆ Adapter classes ease implementation (only override what you need)



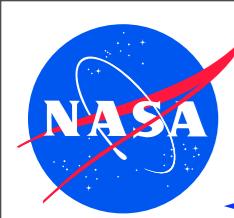


# Basics: Listener Code

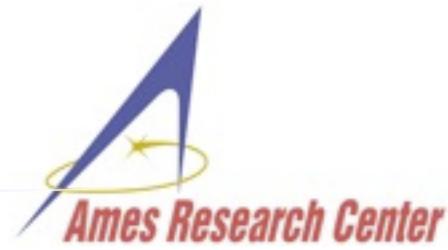


- ◆ extend `gov.nasa.jpf.ListenerAdapter` instead of directly implementing `..Listener`
- ◆ override the notifications you are interested in

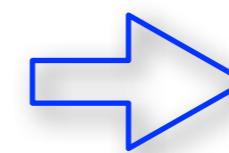
```
public class NonnullChecker extends ListenerAdapter {  
    ...  
    @Override  
    public void executeInstruction (JVM vm){  
        Instruction insn = vm.getLastInstruction();  
        ...  
        if (insn instanceof ARETURN) {  
            ...  
            ...  
        }  
    }  
}
```



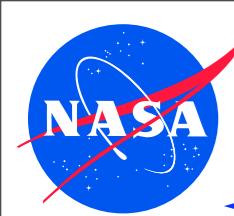
# Basics: Object Attributes



- ◆ user defined (host-VM) objects that can be attached to JPF values
- ◆ attributes travel automatically with values (stack↔stack, stack↔heap)
- ◆ set/processed by extensions (listeners, native peers)
- ◆ good for data flow and data quality properties
- ◆ supports multiple attributes for objects, fields, operands and local vars
- ◆ various APIs
  - ThreadInfo, StackFrame
    - ▶ hasOperandAttr(type), hasLocalAttr(idx,type)
    - ▶ getOperandAttr(type), getLocalAttr(idx,type)
    - ▶ operandAttrIterator(), localAttrIterator(idx)
    - ▶ addOperandAttr(attr), addLocalAttr(idx, attr)
    - ▶ setOperandAttr(attr), setLocalAttr(idx, attr)
    - ▶ ...
  - ElementInfo, Fields
    - ▶ hasObjectAttr(type), hasFieldAttr(type)
    - ▶ getObjectAttr(type), getFieldAttr(field,type)
    - ▶ addObjectAttr(attr), addFieldAttr(field, attr)
    - ▶ setObjectAttr(attr), setFieldAttr(field, attr)
    - ▶ ...



[gov.nasa.jpf.util.ObjectList](http://gov.nasa.jpf.util.ObjectList)

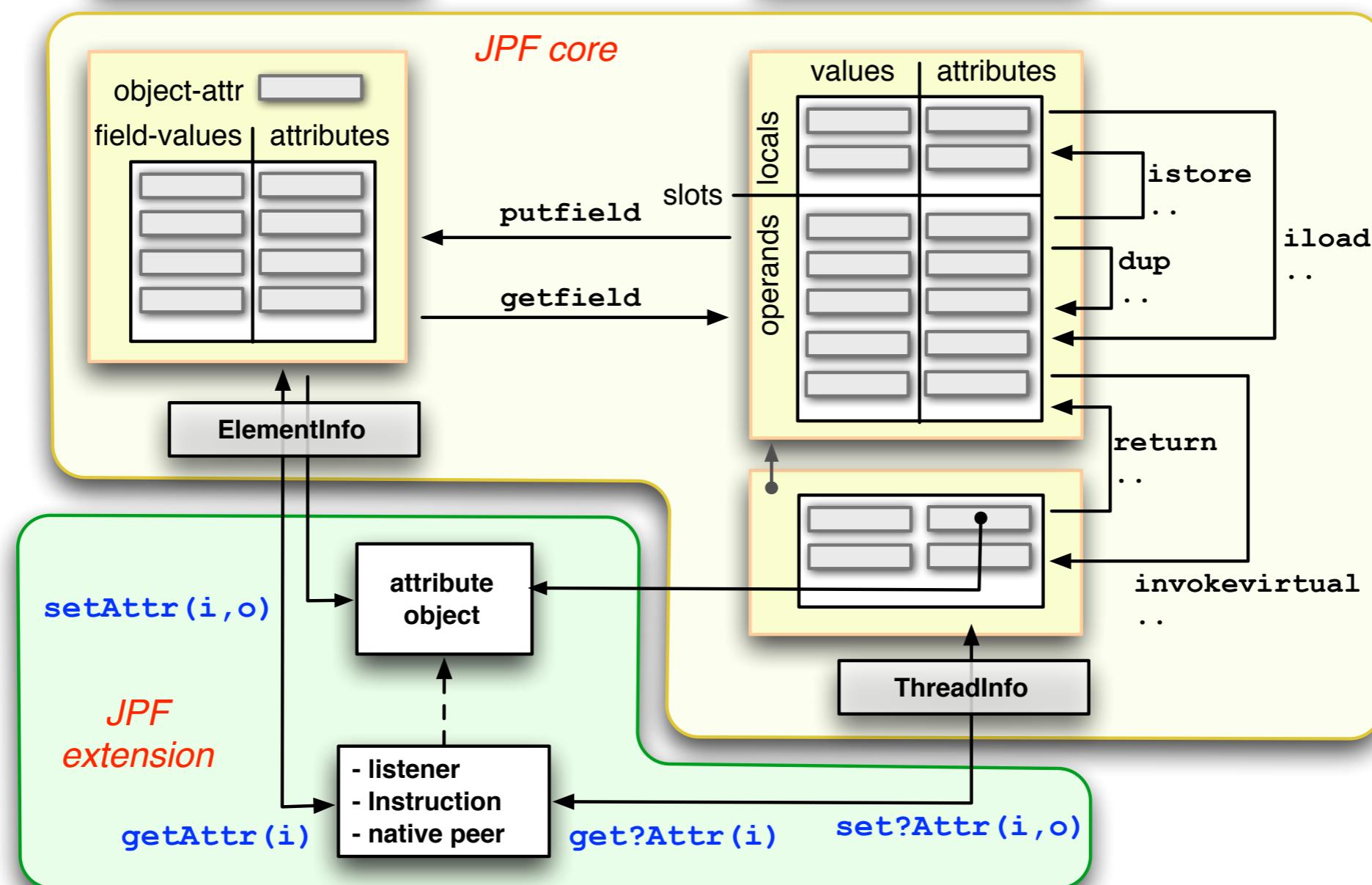


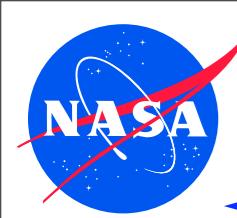
# Basics: Attribute Flow



Fields	StackFrame
int[] values Object[] fieldAttrs Object objectAttr  getIntValue(idx), ... setIntValue(idx, v), ...  ----- getFieldAttr(idx) setFieldAttr(idx,obj) getObjectAttr() setObjectAttr(obj)	int[] locals Object[] localAttr int[] operands Object[] operandAttr  dup(), push(), pop(), ...  ----- getOperandAttr(idx) setOperandAttr(idx,obj) getLocalAttr(idx) setLocalAttr(idx,obj)

*attribute API*



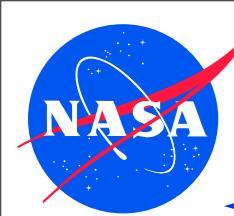


# Basics: Attribute Code



- ◆ attribute class domain/application specific
- ◆ set/processed from listener, native peer or instruction

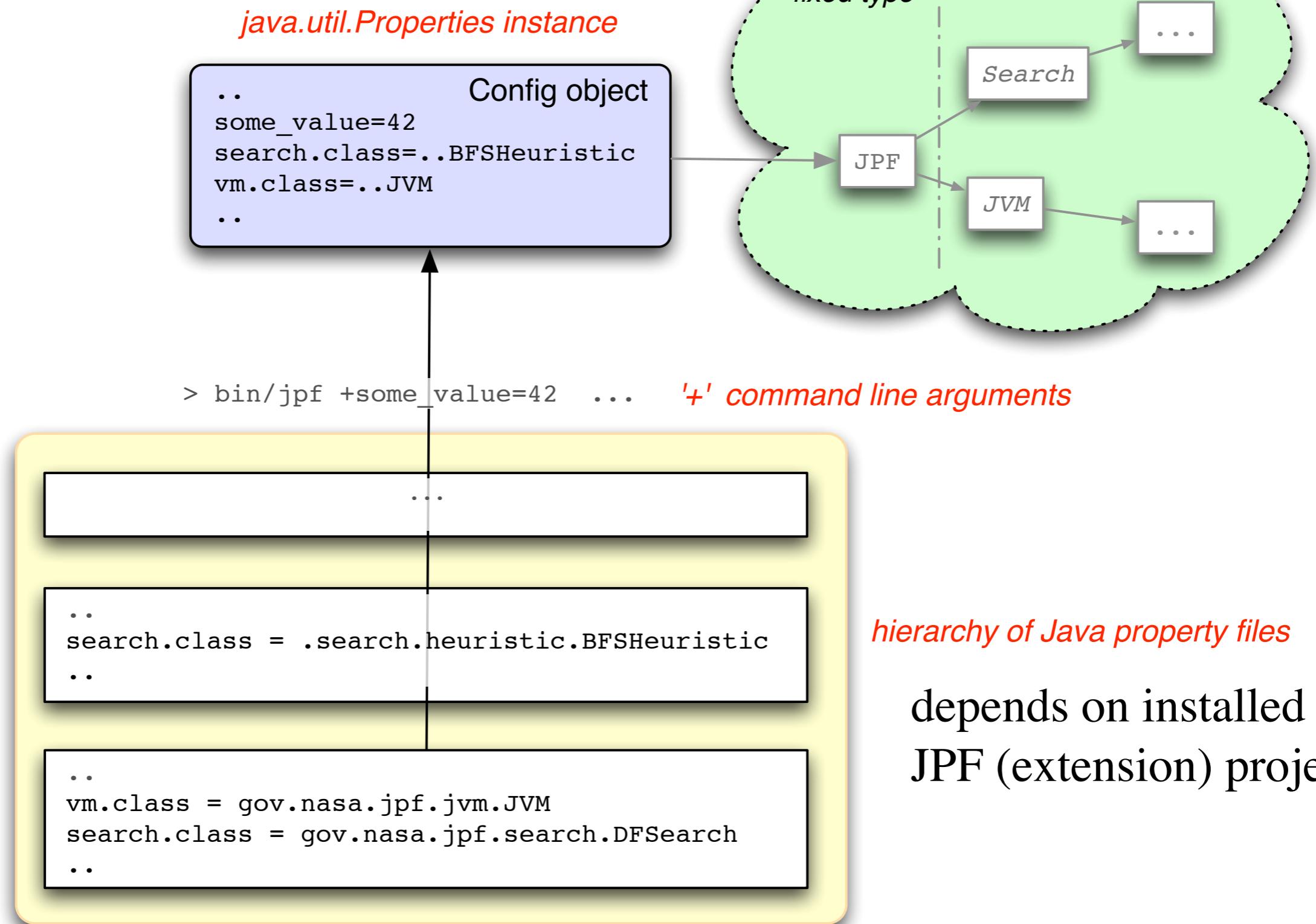
```
public class NumericAttr {  
    ...  
  
public class DSUB extends gov.nasa.jpf.jvm.bytecode.DSUB {  
    ...  
    public Instruction execute (...) {  
        //--- get attributes  
        NumericAttr a1 = ti.getLongOperandAttr(NumericAttr.class);  
        double v1 = ..popDouble();  
        NumericAttr a2 = ..  
        double v2 = ..  
  
        //--- process attributes  
        double r = v2 - v1;  
        NumericAttr ar = a2.subtract(a1);  
        ...  
  
        //--- set attributes  
        ..pushDouble(r);  
        ti.addLongOperandAttr(ar);  
        ...  
    }  
}
```

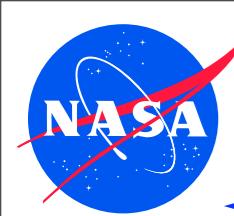


# Basics: Configuration

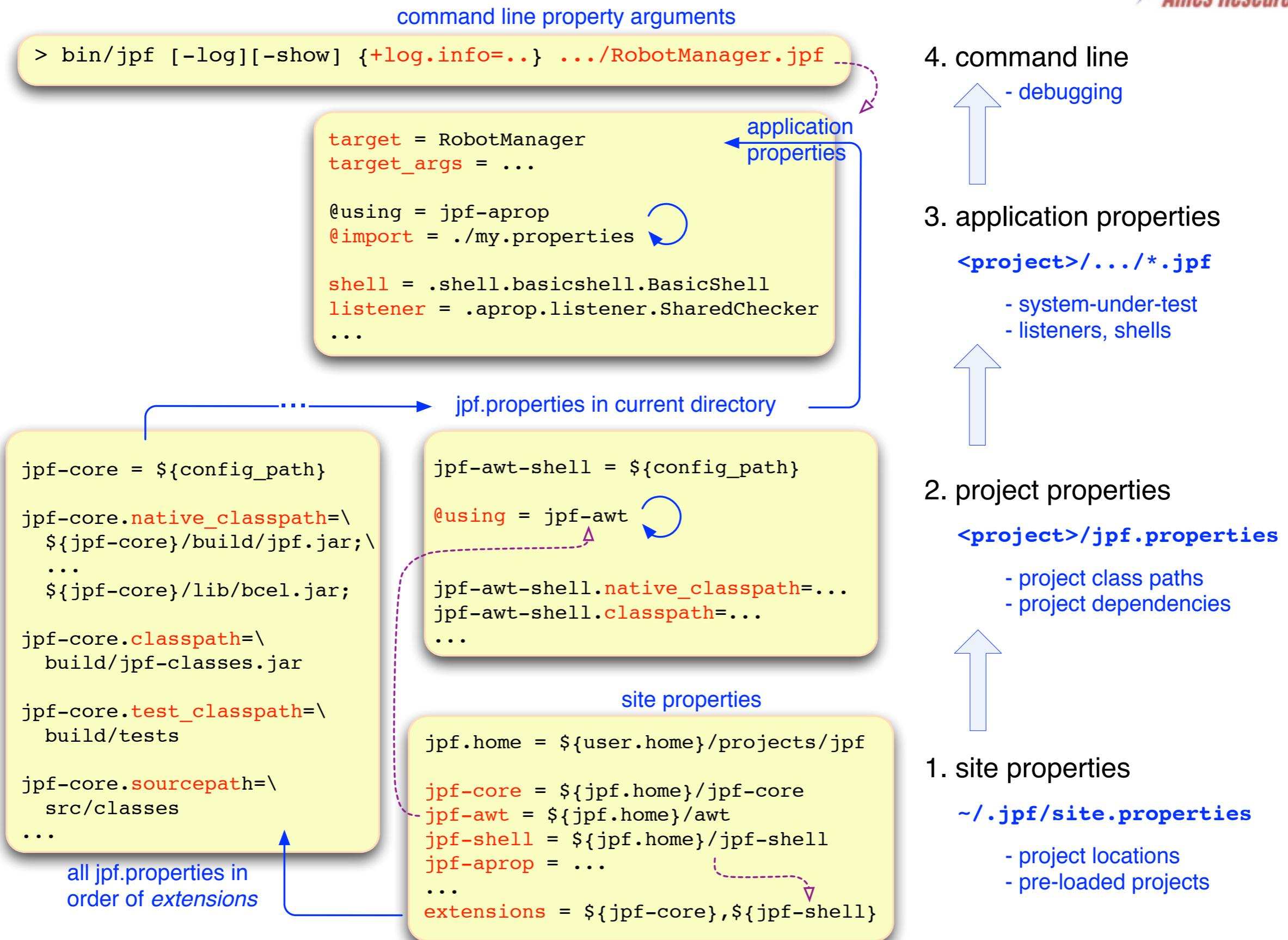
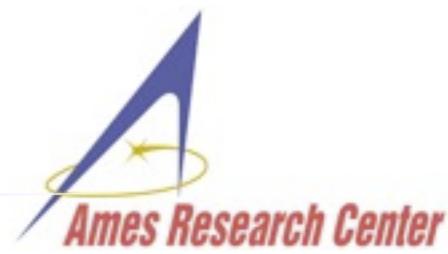


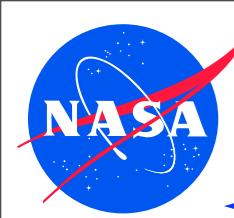
- ◆ open - easy to extend, easy to get lost





# Basics: Configuration Levels

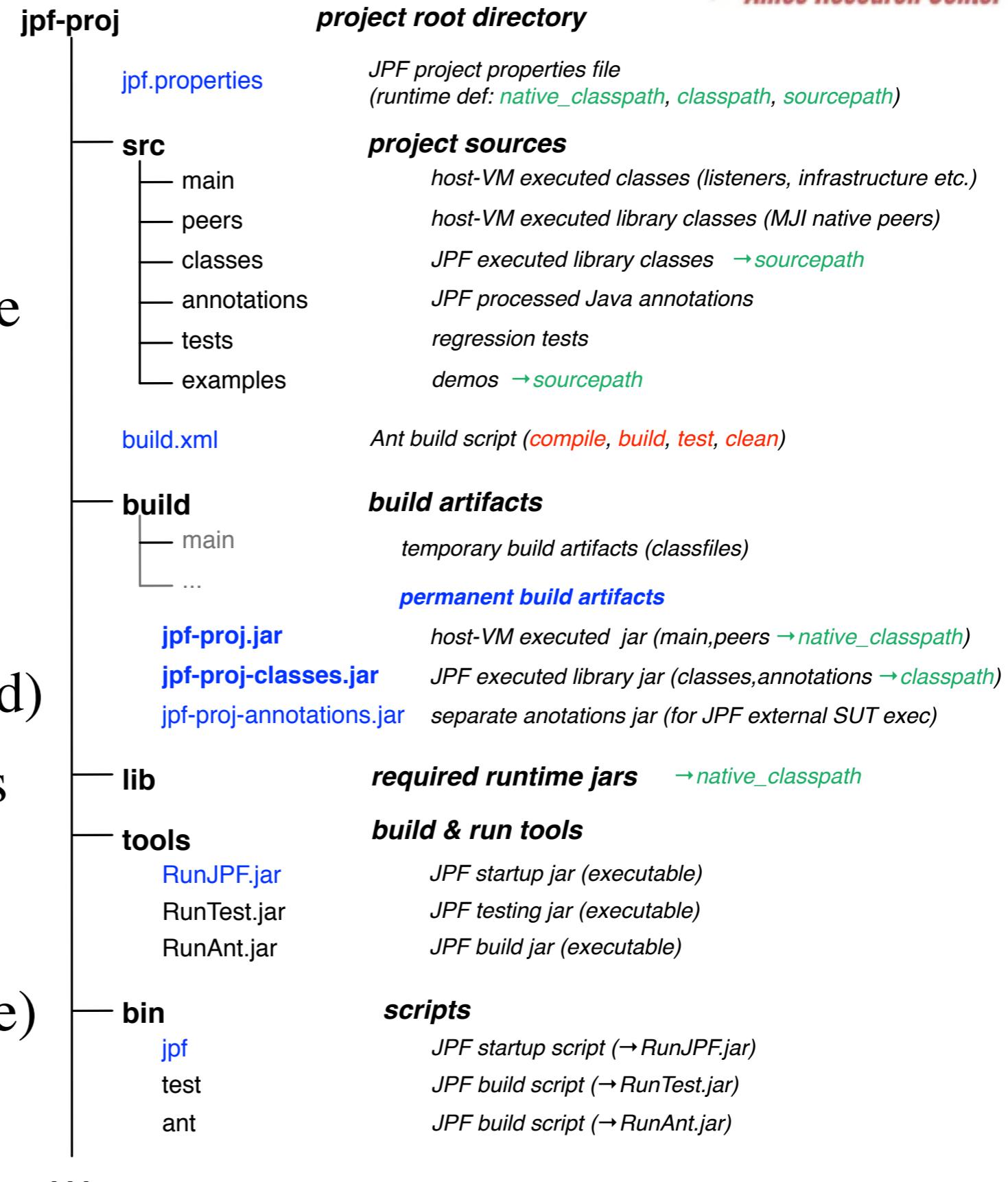


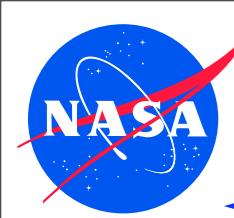


# Basics: JPF Project Layout

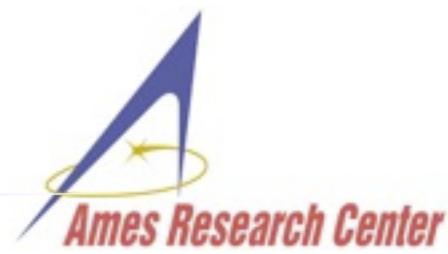


- ◆ all JPF projects share uniform directory layout
- ◆ use ANT based build system
- ◆ use same configuration scheme
- ◆ binary distributions are slices of source distributions (interchangeable)
- ◆ 3rd party tools & libraries can be included (self-contained)
- ◆ projects should have examples and regression test suites
- ◆ projects have out-of-the-box IDE configuration (NB,Eclipse)

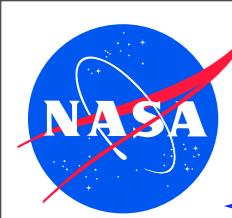




# Basics: Where to get Help - Mailing List



- ◆ <http://groups.google.com/group/java-pathfinder>
- ◆ anyone can join & read, member only post
- ◆ moderate volume (~50 messages/month)
- ◆ subscribe on <http://groups.google.com/group/java-pathfinder/subscribe>



# Basics: Where to get Help - Wiki



<http://babelfish.arc.nasa.gov/trac/jpf>

The screenshot shows the JPF Wiki homepage with several features highlighted:

- Public access:** A red circle highlights the URL in the browser's address bar: <http://babelfish.arc.nasa.gov/trac/jpf/wiki>.
- Ticketing:** A red circle highlights the "View Tickets" and "New Ticket" buttons in the top navigation bar.
- Bug tracking:** A red circle highlights the "Blog" button in the top navigation bar.
- Project blog:** A red circle highlights the sidebar menu under "JPFWiki – Welcome Page".
- Hierarchical navigation menu:** A red circle highlights the "Table of Context" item in the sidebar menu.

**Latest JPF News**

02/14/2010	<a href="#">ISSTA 2010 Tutorial on Automated Testing with Java PathFinder announced</a>
02/12/2010	Call for Google Summer of Code 2010 project proposals out on <a href="#">JPF Google group</a>
01/30/2010	<a href="#">JPF Google group replaces old mailing lists</a>
01/12/2010	<a href="#">Fujitsu press announcement</a> released about using and extending Symbolic Pathfinder ( <a href="#">projects/jpf-symbc</a> ) for comprehensive testing of Java web applications
09/02/2009	JPF server on <a href="http://babelfish.arc.nasa.gov/trac/jpf">http://babelfish.arc.nasa.gov/trac/jpf</a> goes live, featuring the JPFWiki and separate Mercurial repositories for JPF core and extension projects
07/22/2009	JPF wins the 2009 "Outstanding Technology Development Award" of the Federal Laboratory Consortium (FLC), Far West Division

**Welcome to the JPF Wiki**

This is the main page for Java™ Pathfinder, or "**JPF**" as we call it from here. JPF is a highly customizable execution environment for verification of Java™ bytecode programs. The system was developed at the [NASA Ames Research Center](#), open sourced in 2005, and is freely available from this server under the [NOSA 1.3](#) license.

The JPFWiki is our primary source of documentation. It is divided into the following sections (which you will always see in the TOC menu to the right):

- public read access
- edit for account holders (also non-NASA)

bug tracking

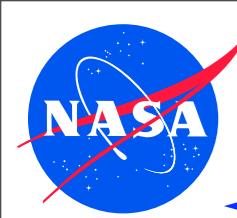
- Trac ticket system

project blog

- announcements
- important changes

hierarchical navigation menu

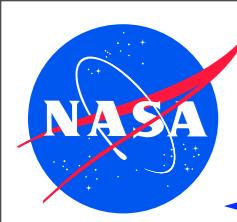
- intro
- installation
- user docu
- developer docu
- extension projects



# Basics: Recap



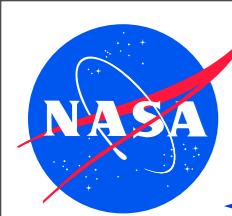
- ◆ the core of JPF is a VM running on top of a Java VM
  - JPF executes the bytecode of the test application
- ◆ JPF is not a monolithic tool, it is highly configurable / extensible
  - ChoiceGenerators
  - Listeners
  - Attribute Objects
  - ...
- ◆ extensions should follow JPF conventions
  - directory layout
  - configuration mechanism



# Basics: Background



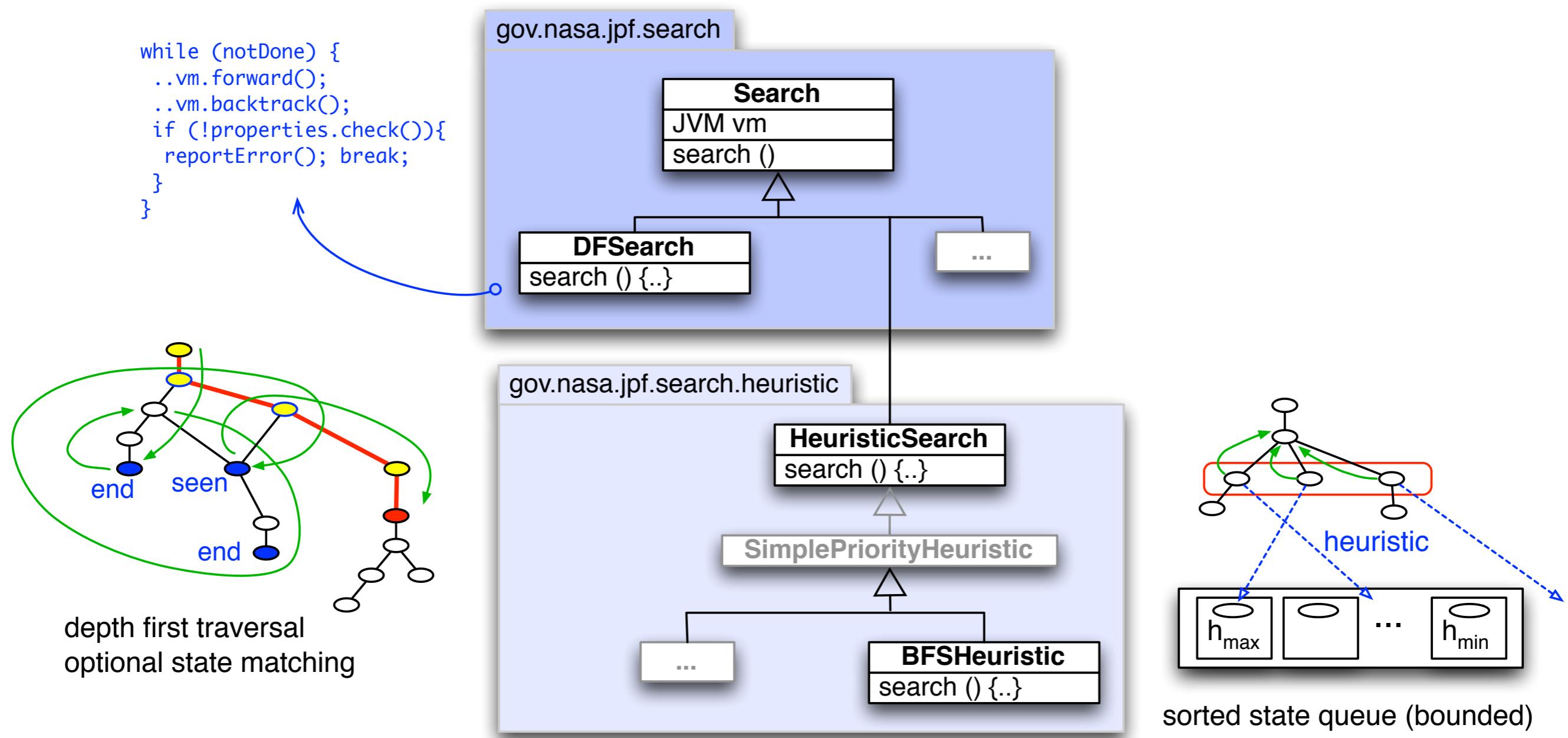
- ◆ ...time permitting

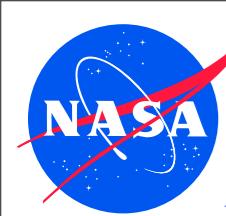


# Basics: Search Policies

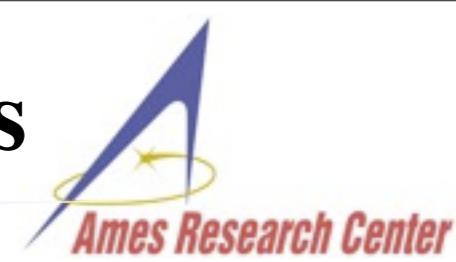


- state explosion mitigation: search the interesting state space part first (“get to the bug early, before running out of memory”)
- Search instances encapsulate (configurable) search policies

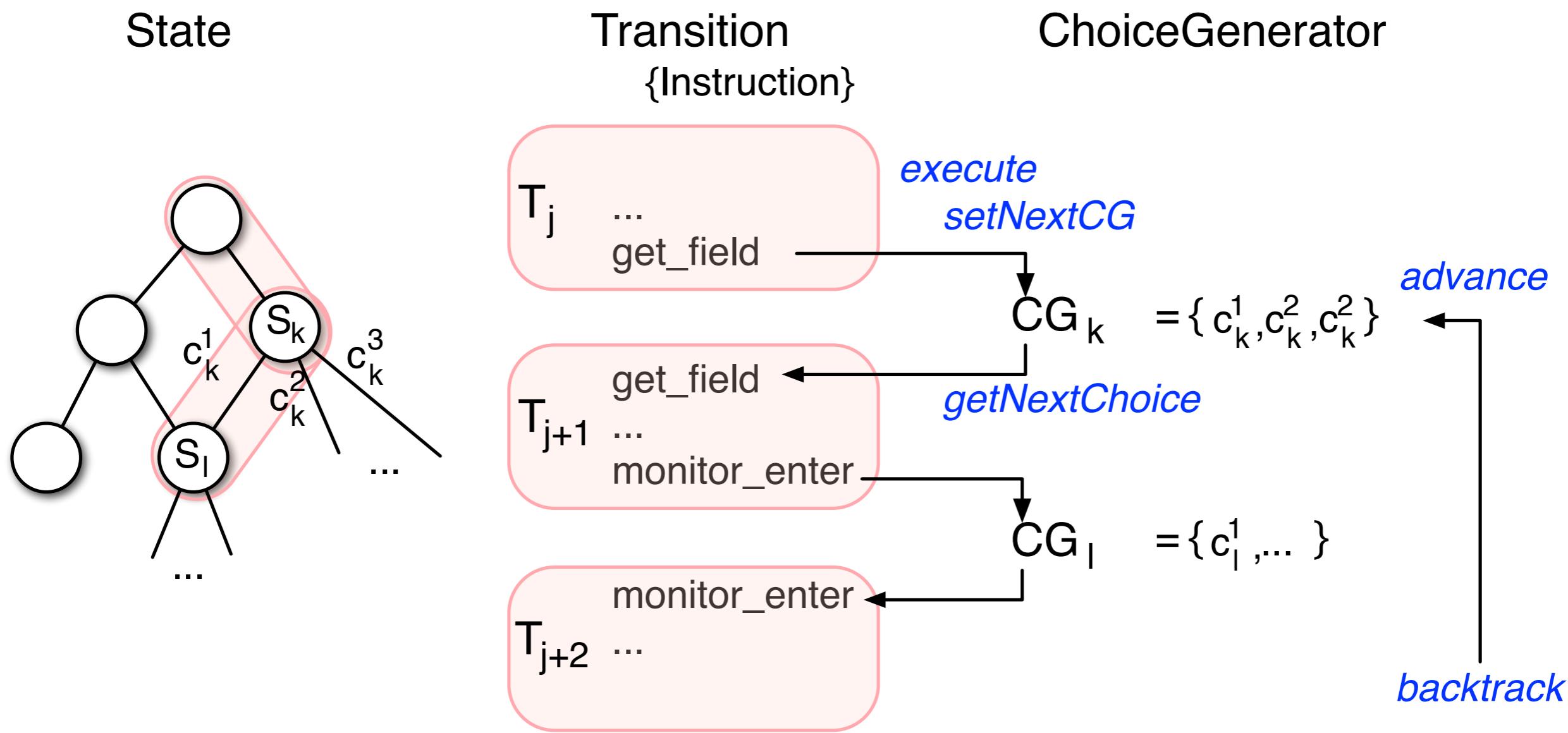


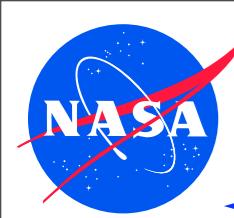


# Basics: ChoiceGenerators, Transitions, States

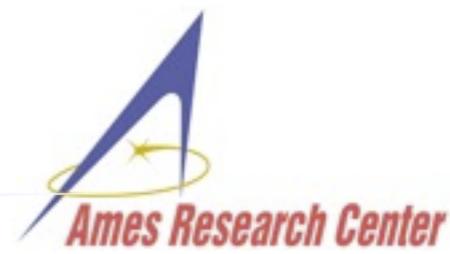


- ◆ transitions begin with a choice and extend until the next ChoiceGenerator (CG) is set (by instruction, native peer or listener)
- ◆ *advance* - positions the CG on the next unprocessed choice (if any)
- ◆ *backtrack* - goes up to the next CG with unprocessed choices

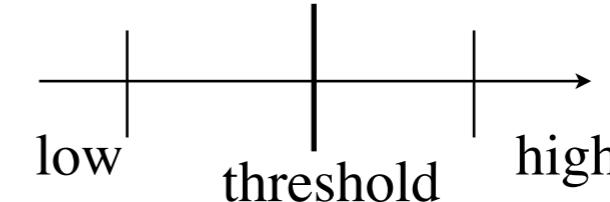




# Basics: ChoiceGenerator Example



```
public class DoubleThresholdGenerator extends DoubleChoiceGenerator {  
    double[] values = new double[3];  
    int count;
```



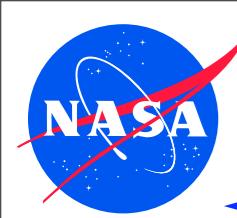
```
public DoubleThresholdGenerator(Config conf, String id) {  
    super(id);  
    values[0] = conf.getDouble(id + ".low");  
    values[1] = conf.getDouble(id + ".threshold");  
    values[2] = conf.getDouble(id + ".high");  
    count = -1;  
}
```

```
public boolean hasMoreChoices () {  
    return !isDone && (count < 2);  
}  
public void advance () {  
    if (count < 2) count++;  
}  
public Double getNextChoice () {  
    return (count >+ 0) ? new Double(values[count])  
                      : new Double(values[0]);  
}
```

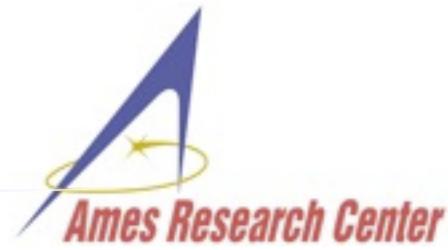
```
public int getTotalNumberOfChoices () { return 3; }  
public int getProcessedNumberOfChoices () { return count + 1; }  
public void reset () { count = -1; }
```

*initialization*

*enumeration*

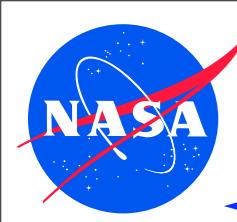


# Basics: Listener Example

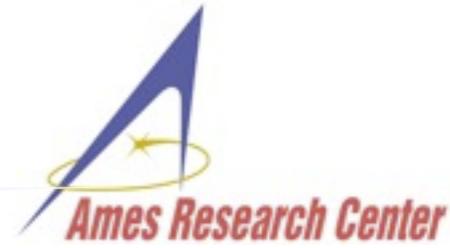


- ◆ extend ListenerAdapter instead of implementing Listener (interface might grow)

```
public class NonnullChecker extends ListenerAdapter {  
    ...  
    public void executeInstruction (JVM vm){  
  
        Instruction insn = vm.getLastInstruction();  
        ThreadInfo ti = vm.getLastThreadInfo();  
  
        if (insn instanceof ARETURN) { // check @NonNull method returns  
            ARETURN areturn = (ARETURN)insn;  
            MethodInfo mi = insn.getMethodInfo();  
  
            if (areturn.getReturnValue(ti) == null) {  
                if (mi.getAnnotation("javax.annotation.NonNull") != null) {  
                    Instruction nextPc =  
                        ti.createAndThrowException("java.lang.AssertionError",  
                            "null return from @NonNull method: " + mi.getCompleteName());  
                    ti.setNextPC(nextPc);  
                    return;  
                }  
            }  
        }  
    }  
}
```

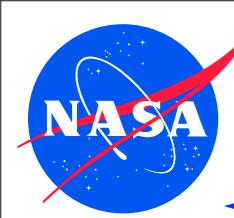


# Basics: Listener Patterns



- ◆ cascaded `instanceof` bad if many alternatives, use Visitor pattern

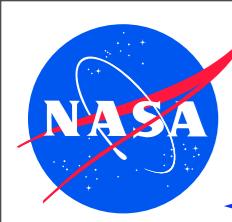
```
public class MyListener extends gov.nasa.jpf.ListenerAdapter {  
    class InsnDispatcher  
        extends gov.nasa.jpf.jvm bytecode InstructionVisitorAdapter {  
            ...  
            void visit (ARETURN insn){  
                ThreadInfo ti = vm.getLastThreadInfo();  
                .. if (areturn.getReturnValue(ti) ..  
            }  
        }  
        ...  
        VM vm;  
        InsnDispatcher dispatcher;  
  
    public MyListener(Config conf){ .. dispatcher = new InsnDispatcher(); ..}  
  
    public void executeInstruction (JVM vm){  
        this.vm = vm;  
        vm.getLastInstruction().accept(dispatcher);  
    }  
}
```



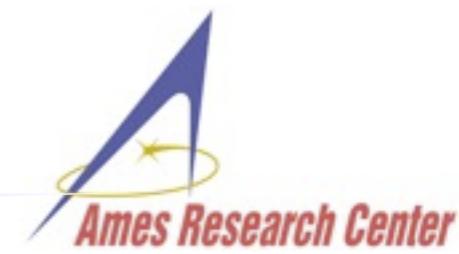
# Basics info: Native Peers



- ◆ what to do with (host-VM) native methods?
  - state lives outside of Java (files, network, windows, ..)
  - how can we backtrack?
- ◆ solution: intercept native method calls and replace with calls to *NativePeer* methods that are executed by host VM
- ◆ association done at class load time
  
- ◆ native peer methods can be used for more than native code:
  - atomic
  - not automatically state tracked
  - can directly interact with JPF (ChoiceGenerator creation etc.)
- ◆ can be venerable optimization
- ◆ main challenge is to translate between different object models: **MJI**



# Basics info: MJI - Model Java Interface



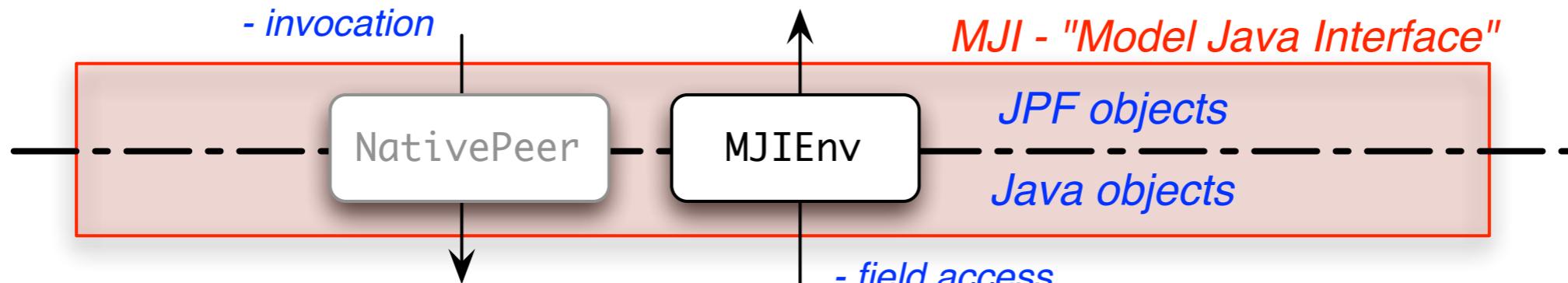
- ◆ transfer from JPF executed code into JVM executed code

```
package x.y.z;  
class MyClass {  
    ..  
    native String foo (int i, String s);  
}
```

*"Model" Class*

*JPF executed*

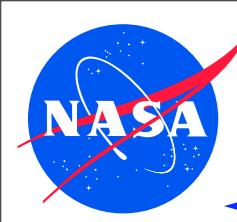
- method lookup
- parameter conversion
- invocation



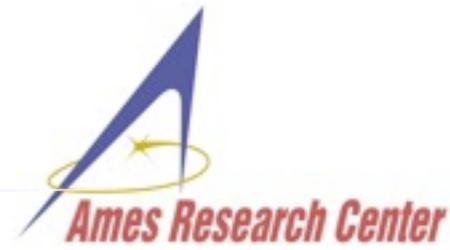
```
class JPF_x_y_z_MyClass {  
    public static int  
        foo__ILjava_lang_String_2__Ljava_lang_String_2 (MJIEnv env, int objRef,  
                                                int i, int sRef) {  
    String s = env.getStringObject(sRef);  
    ..  
    int ref = env.newString(..);  
    return ref;  
}  
}
```

*host VM executed*

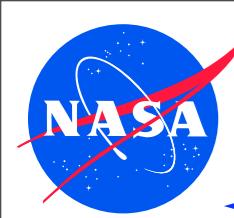
*"NativePeer" Class*



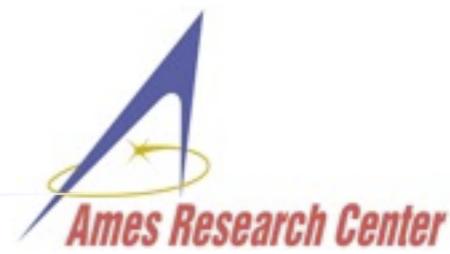
# Basics info: Native Peer Example



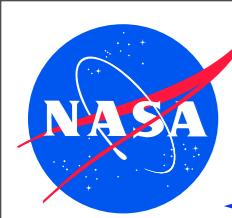
```
public class JPF_java_lang_String {  
    ...  
    public static int indexOf__I__I (MJIEnv env, int objref, int c) {  
        int vref = env.getReferenceField(objref, "value");  
        int off = env.getIntField(objref, "offset");  
        int len = env.getIntField(objref, "count");  
  
        for (int i=0, j=off; i<len; i++, j++) {  
            if ((int)env.getCharArrayElement(vref, j) == c) return i;  
        }  
        return -1;  
    }  
  
    public static int toCharArray____3C (MJIEnv env, int objref){  
        ...  
        int cref = env.newCharArray(len);  
        for (int i=0, j=off; i<len; i++, j++) {  
            env.setCharArrayElement(cref, i, env.getCharArrayElement(vref, j));  
        }  
        return cref;  
    }  
  
    public static boolean matches__Ljava_lang_String_2__Z (MJIEnv env, int objRef,  
                                                       int regexRef){  
        String s = env.getStringObject(objRef);  
        String r = env.getStringObject(regexRef);  
        return s.matches(r);  
    }  
}
```



# Basics info: Instruction Factories



- ◆ JVM is (mostly) agnostic to what `Instruction.execute()` does
- ◆ concrete `Instruction` class hierarchy represents execution semantics
- ◆ can be configured at startup time and replaced at runtime (`MethodInfo` keeps code as replaceable `Instruction` array)
- ◆ JVM uses a configured `InstructionFactory` class to delegate instantiation of instruction objects



# Basics info: InstructionFactory Motivation



- ◆ provide alternative Instruction classes for relevant bytecodes
- ◆ create & configure InstructionFactory that instantiates them
- ◆ overflow example:

JPF configuration

```
vmInsn_factory.class =  
    numeric.NumericInstructionFactory
```

```
class IADD extends Instruction {  
    Instruction execute (... ThreadInfo ti) {  
        int v1 = ti.pop();  
        int v2 = ti.pop();  
        int res = v1 + v2;  
  
        if ((v1>0 && v2>0 && res<=0) ...  
            throw ArithmeticException..
```

compiler

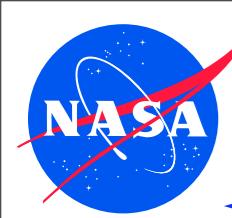
```
...  
[20] iinc  
[21] goto 10  
[10] iload_4  
[11] bipush  
[12] if_icmpge 22  
[13] iload_3  
[14] iload_2  
[15] iadd  
...
```

class loading

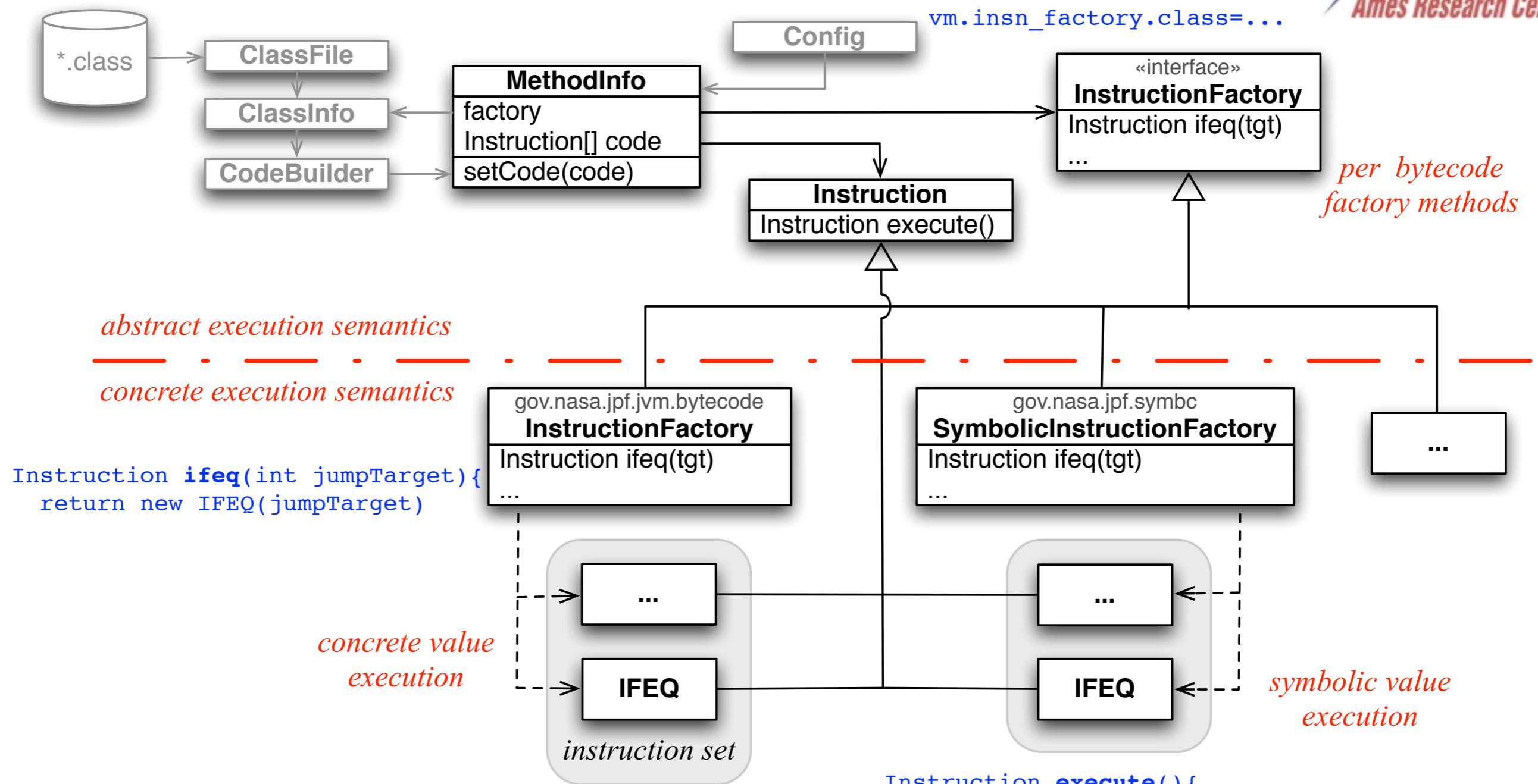
code execution  
(by JPF)

```
void notSoObvious(int x){  
    int a = x*50;  
    int b = 19437583;  
    int c = a;  
  
    for (int k=0; k<100; k++){  
        c += b;  
        System.out.println(c);  
    }  
}  
...  
notSoObvious( 21474836);
```





# Basics: Instruction Factory Implementation



```
Instruction execute (){  
    cond = popCondition()  
    if (cond)  
        return jumpTarget  
    else  
        return getNextInsn()
```

```
Instruction execute(){..  
    if (!firstStepInsn()) {  
        setNextCG(new PCChoiceGenerator())  
        return this  
    }  
    popCondition() // not interested  
    cond = getCG().getNextChoice()  
    if (cond) {...  
        return jumpTarget  
    } else {...  
        return getNextInsn()  
    }  
}
```